WEST Search History

Hide Items | Restore | Clear | Cancel

DATE: Thursday, July 22, 2004

Hide?	Set Name	Query	Hit Count
		DB=PGPB,USPT,EPAB,JPAB,DWPI,TDBD; PLUR=YES; OP=ADJ	
☐	L8	L3 and portal	2
		DB=USPT,PGPB,JPAB,EPAB,DWPI,TDBD; PLUR=YES; OP=ADJ	
☐	L7	(constant pool) near2 redundant	3
		DB=USPT; PLUR=YES; OP=ADJ	
☐	L6	L5	14
		DB=USPT,PGPB,JPAB,EPAB,DWPI,TDBD; PLUR=YES; OP=ADJ	
☐	L5	L3 and bytecode	23
		DB=USPT; PLUR=YES; OP=ADJ	
☐	L4	L3	18
		DB=USPT,PGPB,JPAB,EPAB,DWPI,TDBD; PLUR=YES; OP=ADJ	
☐	L3	L2 and redundant	33
☐	L2	class file and constant pool	181
		DB=PGPB,USPT,USOC; PLUR=YES; OP=ADJ	
☐	L1	717/100-123,162-167.ccls.	2688

END OF SEARCH HISTORY

Hit List

Clear | Generate Collection | Print | Fwd Refs | Bkwd Refs | Generate OACS

Search Results - Record(s) 1 through 3 of 3 returned.

☐ 1. Document ID: US 20040015852 A1

L7: Entry 1 of 3 File: PGPB Jan 22, 2004

PGPUB-DOCUMENT-NUMBER: 20040015852
PGPUB-FILING-TYPE: new
DOCUMENT-IDENTIFIER: US 20040015852 A1

TITLE: System and method for transforming object code

PUBLICATION-DATE: January 22, 2004

INVENTOR-INFORMATION:

NAME	CITY	STATE	COUNTRY	RULE-47
Swetland, Brian	Mountain View	CA	US	

US-CL-CURRENT: 717/118; 717/126

ABSTRACT:

A method comprising: converting bytecodes into a graph of jop objects to track where jump operations pointed before modification of the bytecodes; adjusting constant pool references from local to global numbers based on the graph; and combining the bytecodes into a bundle.

Full | Title | Citation | Front | Review | Classification | Date | Reference | Sequences | Attachments | Claims | KWIC | Draw. Desc | Ir

☐ 2. Document ID: US 20030088851 A1

L7: Entry 2 of 3 File: PGPB May 8, 2003

PGPUB-DOCUMENT-NUMBER: 20030088851
PGPUB-FILING-TYPE: new
DOCUMENT-IDENTIFIER: US 20030088851 A1

TITLE: Method and system for global constant management for memory

PUBLICATION-DATE: May 8, 2003

INVENTOR-INFORMATION:

NAME	CITY	STATE	COUNTRY	RULE-47
Harscoet, Philippe	Santa Clara	CA	US	

US-CL-CURRENT: 717/116; 717/118

ABSTRACT:

Method and system for global constant management. A method of operating a computer is described in which, for data structures and a set of data structures, the date structure is received from a first memory. The data structure includes one or more sets of instructions and a set of one or more constants. The data structure is stored in a second memory. If constants from the data structure have not been stored in other data structures in the second memory, other than the first data structure, then constants in the data structures are stored in data structures in second memory. The constants from the first data structure in the second memory are replaced with links to respective other data structures in the second memory. In one example system, the data structure from the first memory comprises a Java class, and the sets of instructions comprise Java methods. The constants from the data structure in the first memory may comprise a constant pool.

Full | Title | Citation | Front | Review | Classification | Date | Reference | Sequences | Attachments | Claims | KWIC | Draw. Desc | Ir

---

☐ 3. Document ID: US 20020170047 A1

L7: Entry 3 of 3 File: PGPB Nov 14, 2002

PGPUB-DOCUMENT-NUMBER: 20020170047
PGPUB-FILING-TYPE: new
DOCUMENT-IDENTIFIER: US 20020170047 A1

TITLE: System and method for transforming object code

PUBLICATION-DATE: November 14, 2002

INVENTOR-INFORMATION:

NAME	CITY	STATE	COUNTRY	RULE-47
Swetland, Brian	Mountain View	CA	US	

US-CL-CURRENT: 717/162

ABSTRACT:

A unified programming object is described comprising: a shared constant pool comprising global constant pool entries mapped from local constant pool entries of two or more class files; and a plurality of object code copied from the two or more class files to the unified programming object and identified by the global constant pool entries.

Full | Title | Citation | Front | Review | Classification | Date | Reference | Sequences | Attachments | Claims | KWIC | Draw. Desc | Ir

---

Clear | Generate Collection | Print | Fwd Refs | Bkwd Refs | Generate OACS

Terms	Documents

Terms used **constant pool redundant class file**

Found **22** of **139,988**

Sort results by: relevance

Display results: expanded form

Save results to a Binder

Search Tips

☐ Open results in a new window

Try an Advanced Search

Try this search in The ACM Guide

Results 1 - 20 of 22

Result page: **1** 2 next

Relevance scale ☐ ▬ ▬ ▬ ■

1 Practical extraction techniques for Java

Frank Tip, Peter F. Sweeney, Chris Laffra, Aldo Eisma, David Streeter

November 2002 **ACM Transactions on Programming Languages and Systems (TOPLAS)**, Volume 24 Issue 6

Full text available: pdf(1.01 MB) Additional Information: full citation, abstract, references, citings, index terms, review

Reducing application size is important for software that is distributed via the internet, in order to keep download times manageable, and in the domain of embedded systems, where applications are often stored in (Read-Only or Flash) memory. This paper explores extraction techniques such as the removal of unreachable methods and redundant fields, inlining of method calls, and transformation of the class hierarchy for reducing application size. We implemented a number of extraction techniques in < ...

Keywords: Application extraction, call graph construction, class hierarchy transformation, packaging, whole-program analysis

2 JAZZ: an efficient compressed format for Java archive files

Quetzalcoatl Bradley, R. Nigel Horspool, Jan Vitek

November 1998 **Proceedings of the 1998 conference of the Centre for Advanced Studies on Collaborative research**

Full text available: pdf(73.54 KB) Additional Information: full citation, abstract, references, index terms

The Jazz file format is intended to be a replacement for the JAR file format when used for storage and distribution of Java programs. A Jazz file is compressed to a degree that far exceeds what is possible with a JAR file. The smaller size of the Jazz format permits faster transmission speeds over a network and has the additional benefit of conserving disk storage. The compression is achieved as a combination of different data compression methods, adapted to suit the characteristics of collectio ...

3 Techniques for obtaining high performance in Java programs

Iffat H. Kazi, Howard H. Chen, Berdenia Stanley, David J. Lilja

September 2000 **ACM Computing Surveys (CSUR)**, Volume 32 Issue 3

Full text available: pdf(816.13 KB) Additional Information: full citation, abstract, references, citings, index terms

This survey describes research directions in techniques to improve the performance of programs written in the Java programming language. The standard technique for Java

execution is interpretation, which provides for extensive portability of programs. A Java interpreter dynamically executes Java bytecodes, which comprise the instruction set of the Java Virtual Machine (JVM). Execution time performance of Java programs can be improved through compilation, possibly at the expense of portabili ...

Keywords: Java, Java virtual machine, bytecode-to-source translators, direct compilers, dynamic compilation, interpreters, just-in-time compilers

4 Virtual machine support for dynamic join points

Christoph Bockisch, Michael Haupt, Mira Mezini, Klaus Ostermann

March 2004 **Proceedings of the 3rd international conference on Aspect-oriented software development**

Full text available: pdf(1.19 MB) Additional Information: full citation, abstract, references, citings

A widespread implementation approach for the join point mechanism of aspect-oriented languages is to instrument areas in code that match the static part of pointcut designators, inserting dynamic checks for that part of matching that depends on run-time conditions, if needed. For performance reasons, such dynamic checks should be avoided whenever possible. One way to do so is to postpone weaving of advice calls until run-time, when conditions determining the emergence of join points hold. This c ...

5 Compressing Java class files

William Pugh

May 1999 **ACM SIGPLAN Notices , Proceedings of the ACM SIGPLAN 1999 conference on Programming language design and implementation**, Volume 34 Issue 5

Full text available: pdf(1.44 MB) Additional Information: full citation, abstract, references, citings, index terms

Java class files are often distributed as jar files, which are collections of individually compressed class files (and possibility other files). Jar files are typically about 1/2 the size of the original class files due to compression. I have developed a wire-code format for collections of Java class files. This format is typically 1/2 to 1/5 of the size of the corresponding compressed jar file (1/4 to 1/10 the size of the original class files).

6 Practical experience with an application extractor for Java

Frank Tip, Chris Laffra, Peter F. Sweeney, David Streeter

October 1999 **ACM SIGPLAN Notices , Proceedings of the 14th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications**, Volume 34 Issue 10

Full text available: pdf(2.31 MB) Additional Information: full citation, abstract, references, citings, index terms

Java programs are routinely transmitted over low-bandwidth network connections as compressed class file archives (i.e., zip files and jar files). Since archive size is directly proportional to download time, it is desirable for applications to be as small as possible. This paper is concerned with the use of program transformations such as removal of dead methods and fields, inlining of method calls, and simplification of the class hierarchy for reducing application size. Such "extract ...

7 Soot - a Java bytecode optimization framework

Raja Vallée-Rai, Phong Co, Etienne Gagnon, Laurie Hendren, Patrick Lam, Vijay Sundaresan

November 1999 **Proceedings of the 1999 conference of the Centre for Advanced Studies on Collaborative research**

Full text available: pdf(79.70 KB) Additional Information: full citation, abstract, references, citings, index terms

This paper presents Soot, a framework for optimizing Java bytecode. The framework is implemented in Java and supports three intermediate representations for representing Java bytecode: Baf, a streamlined representation of bytecode which is simple to manipulate; Jimple, a typed 3-address intermediate representation suitable for optimization; and Grimp, an aggregated version of Jimple suitable for decompilation. We describe the motivation for each representation, and the salient points in translat ...

8 Software techniques for program compaction: Extracting library-based Java applications

Frank Tip, Peter F. Sweeney, Chris Laffra

August 2003 **Communications of the ACM**, Volume 46 Issue 8

Full text available: pdf(235.95 KB) html(28.30 KB) Additional Information: full citation, abstract, references, index terms, review

Reducing the size of Java applications by creating an application extractor.

9 Using production grammars in software testing

Emin Gün Sirer, Brian N. Bershad

December 1999 **ACM SIGPLAN Notices , Proceedings of the 2nd conference on Domain-specific languages**, Volume 35 Issue 1

Full text available: pdf(1.27 MB) Additional Information: full citation, abstract, references, index terms

Extensible typesafe systems, such as Java, rely critically on a large and complex software base for their overall protection and integrity, and are therefore difficult to test and verify. Traditional testing techniques, such as manual test generation and formal verification, are too time consuming, expensive, and imprecise, or work only on abstract models of the implementation and are too simplistic. Consequently, commercial virtual machines deployed so far have exhibited numerous bugs and ...

10 Using annotations to reduce dynamic optimization time

Chandra Krintz, Brad Calder

May 2001 **ACM SIGPLAN Notices , Proceedings of the ACM SIGPLAN 2001 conference on Programming language design and implementation**, Volume 36 Issue 5

Full text available: pdf(1.78 MB) Additional Information: full citation, abstract, references, citings, index terms

Dynamic compilation and optimization are widely used in heterogenous computing environments, in which an intermediate form of the code is compiled to native code during execution. An important trade off exists between the amount of time spent dynamically optimizing the program and the running time of the program. The time to perform dynamic optimizations can cause significant delays during execution and also prohibit performance gains that result from more complex optimization.

11 A framework for optimizing Java using attributes

Patrice Pominville, Feng Qian, Raja Vallée-Rai, Laurie Hendren, Clark Verbrugge

November 2000 **Proceedings of the 2000 conference of the Centre for Advanced Studies on Collaborative research**

Full text available: pdf(314.37 KB) Additional Information: full citation, abstract, references, citings, index terms

This paper presents a framework for supporting the optimization of Java programs using attributes in Java class files. We show how class file attributes may be used to convey both optimization opportunities and profile information to a variety of Java virtual machines including ahead-of-time compilers and just-in-time compilers.We present our work in the context of Soot, a framework that supports the analysis and transformation of Java bytecode (class files)[21, 25, 26]. We demonstrate the frame ...

12 Language-specific make technology for the Java programming language
Mikhail Dmitriev
November 2002 **ACM SIGPLAN Notices , Proceedings of the 17th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications**, Volume 37 Issue 11
Full text available: pdf(238.19 KB) Additional Information: full citation, abstract, references, index terms

Keeping the code of a Java application consistent (code is consistent if all of the project classes can be recompiled together without errors) prevents late linking errors, and thus may significantly improve development turnaround time. In this paper we describe a make technology for the Java programming language, that is based on smart dependency checking, guarantees consistency of the project code, and at the same time reduces the number of source code recompilations to the minimum. After proj ...

Keywords: Java, dependency checking, development turnaround time, make

13 Engineering a customizable intermediate representation
K. Palacz, J. Baker, C. Flack, C. Grothoff, H. Yamauchi, J. Vitek
June 2003 **Proceedings of the 2003 workshop on Interpreters, Virtual Machines and Emulators**
Full text available: pdf(322.87 KB) Additional Information: full citation, abstract, references, citings

The Ovm framework is a set of tools and components for building language runtimes. We present the intermediate representation and software design patterns used throughout the framework. One of the main themes in this work has been to support experimentation with new linguistic constructs and implementation techniques. To this end, framework components were designed to be parametric with respect to the instruction set on which they operate. We argue that our approach eases the task of writing new ...

14 Software watermarking: models and dynamic embeddings
Christian Collberg, Clark Thomborson
January 1999 **Proceedings of the 26th ACM SIGPLAN-SIGACT symposium on Principles of programming languages**
Full text available: pdf(2.19 MB) Additional Information: full citation, references, citings, index terms

15 The apprentice challenge
J. Strother Moore, George Porter
May 2002 **ACM Transactions on Programming Languages and Systems (TOPLAS)**, Volume 24 Issue 3
Full text available: pdf(212.09 KB) Additional Information: full citation, abstract, references, citings, index terms

We describe a mechanically checked proof of a property of a small system of Java programs involving an unbounded number of threads and synchronization, via monitors. We adopt the output of the javac compiler as the semantics and verify the system at the bytecode level under an operational semantics for the JVM. We assume a sequentially consistent memory model and atomicity at the bytecode level. Our operational semantics is expressed in ACL2, a Lisp-based logic of recursive functions. Our proofs ...

Keywords: Java, Java Virtual Machine, mutual exclusion, operational semantics, parallel and distributed computation, theorem proving

16 Sifting out the mud: low level C++ code reuse
Bjorn De Sutter, Bruno De Bus, Koen De Bosschere
November 2002 **ACM SIGPLAN Notices , Proceedings of the 17th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications**, Volume 37 Issue 11

Full text available: pdf(1.35 MB) Additional Information: full citation, abstract, references, citings, index terms

More and more computers are being incorporated in devices where the available amount of memory is limited. This contrasts with the increasing need for additional functionality and the need for rapid application development. While object-oriented programming languages, providing mechanisms such as inheritance and templates, allow fast development of complex applications, they have a detrimental effect on program size. This paper introduces new techniques to reuse the code of whole procedures at t ...

Keywords: code compaction, code size reduction

17 Extracting library-based object-oriented applications
Peter F. Sweeney, Frank Tip
November 2000 **ACM SIGSOFT Software Engineering Notes , Proceedings of the 8th ACM SIGSOFT international symposium on Foundations of software engineering: twenty-first century applications**, Volume 25 Issue 6

Full text available: pdf(1.06 MB) Additional Information: full citation, abstract, references, citings, index terms

In an increasingly popular model of software distribution, software is developed in one computing environment and deployed in other environments by transfer over the internet. Extraction tools perform a static whole-program analysis to determine unused functionality in applications in order to reduce the time required to download applications. We have identified a number of scenarios where extraction tools require information beyond what can be inferred through static analysis: software distr ...

18 Object equality profiling
Darko Marinov, Robert O'Callahan
October 2003 **ACM SIGPLAN Notices , Proceedings of the 18th ACM SIGPLAN conference on Object-oriented programing, systems, languages, and applications**, Volume 38 Issue 11

Full text available: pdf(577.47 KB) Additional Information: full citation, abstract, references, citings, index terms

We present *Object Equality Profiling* (OEP), a new technique for helping programmers discover optimization opportunities in programs. OEP discovers opportunities for replacing a set of equivalent object instances with a single representative object. Such a set represents an opportunity for automatically or manually applying optimizations such as hash consing, heap compression, lazy allocation, object caching, invariant hoisting, and more. To evaluate OEP, we implemented a tool to help prog ...

Keywords: Java language, object equality, object mergeability, profile-guided optimization, profiling, space savings

19 SafeTSA: a type safe and referentially secure mobile-code representation based on static single assignment form
Wolfram Amme, Niall Dalton, Jeffery von Ronne, Michael Franz
May 2001 **ACM SIGPLAN Notices , Proceedings of the ACM SIGPLAN 2001 conference on Programming language design and implementation**, Volume 36 Issue 5

Full text available: pdf(1.35 MB) Additional Information: full citation, references, citings, index terms

20 Manufacturing cheap, resilient, and stealthy opaque constructs

Christian Collberg, Clark Thomborson, Douglas Low

January 1998 **Proceedings of the 25th ACM SIGPLAN-SIGACT symposium on Principles of programming languages**

Full text available: pdf(1.59 MB) Additional Information: full citation, references, citings, index terms

Results 1 - 20 of 22 Result page: **1** 2 next

CiteSeer Find: 'constant pool' AND class file Documents Citations

Searching for **constant pool and class file**.
Restrict to: Header Title Order by: Expected citations Hubs Usage Date Try: Google (CiteSeer) Google (Web) CSB DBLP
51 documents found. **Order: number of citations.**

A Formal Specification of Java Virtual Machine Instructions for.. - Qian (1998) (Correct) (40 citations)
variables, the **class file** format in details, **constant pool** resolution in details and the difference
the legality of accesses of local variables, the **class file** format in details, **constant pool** resolution in
ftp.kestrel.edu/pub/papers/qian/fsjvm-sembk.ps

A Formal Specification of Java Virtual Machine Instructions - Qian (1997) (Correct) (40 citations)
as predefined functions. We do not consider the **constant pool** resolution in details. Instead, we assume
are written in a special file format, called **class file** format. A **class file** contains JVM instructions
www.cis.upenn.edu/~bcpierce/courses/629/papers/Qian-JVMspec.ps.gz

Java is Not Type-Safe - Saraswat (1997) (Correct) (39 citations)
run-time penalites) by fixing the (link-time) **constant pool** resolution process to take into account
1. The problem Let A and A' be two different **classfiles** defining a Java class with the same fully
www.loria.fr/~lliquori/JAVA/bug.ps

A Specification of Java Loading and Bytecode Verification - Goldberg (1998) (Correct) (21 citations)
a class definition can be parsed to yield a **constant pool** and syntactically-correct code for each
a consistent global typing environment and a **class file**, determine if the **class file** is welltyped given
ftp.kestrel.edu/pub/papers/goldberg/Bytecode.ps.gz

Byte Code Engineering - Dahm (1999) (Correct) (18 citations)
Instances of this class basically consist of a **constant pool**, fields, methods, symbolic references to the
and dynamic creation or transformation of Java **class files**. 1 It enables developers to deal with byte
before the JVM finally executes it. Java **class file** Class loader Byte code verifier Interpreter/JIT Byte
ftp.inf.fu-berlin.de/pub/JavaClass/paper.ps.gz

Compressing Java Class Files - Pugh (1999) (Correct) (15 citations)
requires that we renumber entries in the **constant pool**, I exclude any unrecognized attributes (we
are likely to get good compression. But if we pool **constant pool** entries, it is unlikely that we will be
Compressing Java **Class Files** William Pugh Dept. of Computer Science Univ.
www.cs.umd.edu/~pugh/papers/pack.pdf

Byte Code Engineering with the JavaClass API - Dahm (1999) (Correct) (11 citations)
and the version number, followed by the **constant pool**, which can be roughly thought of as the text
Java classes are compiled into portable binary **class files** (called byte code)it is the most convenient
before the JVM finally executes it. Java **class file** Class loader Byte code verifier Interpreter/JIT Byte
ftp.inf.fu-berlin.de/pub/JavaClass/report.ps.gz

JAZZ: An Efficient Compressed Format for Java Archive Files - Bradley, Horspool, Vitek (1998) (Correct) (10 citations)
component of a **class file** is typically the **constant pool**. It is not unusual for the **constant pool** to
suit the characteristics of collections of Java **class files**. 1 Introduction A typical Java application
www.csr.uvic.ca/~nigelh/Publications/jazz.pdf

Toward a Provably-Correct Implementation of the JVM.. - Coglio, Goldberg, Qian (1998) (Correct) (10 citations)
assumptions derived from declarations in the **constant pool**. Furthermore, a constraint problem is
consistency with referenced classes. Because **class files** are loaded dynamically, and because it is
www-dse.doc.ic.ac.uk/~sue/oopsla/goldberg.f.ps

Semantics of Java Byte Code - Bertelsen (1997) (Correct) (10 citations)
Index is the type of indices into the **constant pool** of a **class file**. PC is the type of program
precise specification of the format of Java **class files**. The Java Virtual Machine is a multi-threaded

www.dina.kvl.dk/pub/Staff/Peter.Bertelsen/jvm-semantics.ps.gz

Java Bytecode Compression for Low-End Embedded Systems - Ræder Clausen.. (2000) (Correct) (6 citations)
1998 Pugh 1999]In the Java class format, the **constant pool** comprises most of the space the bytecode have concentrated on reducing the size of Java **class files** for transmission and subsequent execution on by elements in the Method component of this CAP **file. Class**. Describes each of the classes and interfaces
www.daimi.aau.dk/~ups/papers/toplas00.ps.gz

On the Limits of Software Watermarking - Collberg, Thomborson (1998) (Correct) (6 citations)
of the many sections of the **class file** format: **constant pool** table, method table, line number table, etc. distributed to Bob as a collection of Java **class files**. As we shall see, watermarking Java class
www.cs.auckland.ac.nz/~collberg/Research/Publications/CollbergThomborson98e/A4.ps

Byte Code Engineering with the BCEL API - Dahm (2001) (Correct) (5 citations)
and the version number, followed by the **constant pool**, which can be roughly thought of as the text Java classes are compiled into portable binary **class files** (called byte code)it is the most convenient before the JVM finally executes it. Java **class file** Class loader Byte code verifier Interpreter/JIT Byte
ftp.inf.fu-berlin.de/pub/BCEL/report.ps

Formalizing the Java Virtual Machine in Isabelle/HOL - Pusch (1998) (Correct) (5 citations)
.6 3.6 The **Constant Pool** .
theorem prover Isabelle/HOL. We formalize the **class file** format and give an operational semantics for a
ftp.leo.org/pub/comp/doc/techreports/tum/informatik/report/1998/TUM-I9816.ps.gz

Java Bytecode Compression For Embedded Systems - Ræder Clausen, Schultz.. (1998) (Correct) (5 citations)
information and most, if not all, of the **constant pool**, all that remains is the actual bytecode that used approach for reducing the size of Java **class files** is to remove all non-essential information.
ftp.irisa.fr/techreports/1998/PI-1213.ps.gz

Dynamic semantics of Java byte-code - Bertelsen (1998) (Correct) (4 citations)
its method implementations, and its **constant pool**. The **constant pool** holds string literals, describe the following aspects of the JVM: **class file** verification: what checks can or must be done
ftp.dina.kvl.dk/pub/Staff/Peter.Bertelsen/jvm-dyn-sem.ps.gz

A Formal Introduction to the Compilation of Java - Diehl (1998) (Correct) (4 citations)
class, then #c#KF M P CS#where **Constant Pool** #K :In the **constant pool** there are entries of the static information contained in **class files**, and we discuss only the compilationof a
www.cis.ksu.edu/~hatcliff/605/XC/java-comp.pdf.gz

Compiling SML to Java Bytecode - Bertelsen (1998) (Correct) (3 citations)
File 34 3.5 Code Emission 35 3.5.1 Building the **Constant Pool** 37 3.5.2 Label Resolution 37 3.6 Auxiliary 31 3.3 Abstractions 33 3.4 Generating A **Class File** 34 3.5 Code Emission 35 3.5.1 Building the
ftp.dina.kvl.dk/pub/Staff/Peter.Bertelsen/sml2jvm.ps.gz

First 20 documents Next 20

Try your query at: Google (CiteSeer) Google (Web) CSB DBLP